

# Engineering Web Service - Agent Integration\*

Dominic Greenwood and Monique Calisti  
Whitestein Technologies AG  
Zurich, Switzerland  
E-mail: {dgr, mca}@whitestein.com

**Abstract** – *Web services are fast emerging as the dominant means for connecting remotely executing programs via well established internet protocols and commonly used machine readable representations.*

*Software agents are now increasingly used in commercial applications to solve complex engineering problems, and these applications often expose or make use of Web services.*

*As such, this paper presents a Gateway architecture for connecting software agents and Web services in a transparent manner with fully automatic operation. This Gateway allows Web services to invoke agent services and vice versa by translating message encodings and service descriptions between the two technologies. We also address how software agents offer the opportunity to introduce new modalities in the ways Web services are used and manipulated, including redirection, aggregation, integration and administration.*

## 1 Introduction

Today, business interactions are increasingly triggering and relying upon the use of a variety of electronic and information technologies. In this context, *Web services* are conceived as an essential component for promoting interoperability of business processes and *software agents* as a key enabling technology for such processes to be dynamically discovered, combined and executed [16].

The work reported in this paper is motivated by the recognition that software agent platforms, and particularly JADE<sup>1</sup> as one of the most widely deployed open source agent systems [19], should have the intrinsic capability of seamlessly invoking Web services and allowing Web service clients to make use of agent services. Any agent service to be exposed for consumption by Web services should be invocable using basic request-response operations as this is the communication mode of typical Web services.

After a short introduction on background principles and technologies, including an overview of the most relevant works, the central part of the paper focuses on the *Web Service Integration Gateway Service*, WSIGS. This component aims at providing transparent bidirectional access from/to Web services to/from agent-based services (more details in Section 3). After a presentation of the main WSIGS archi-

tectural components, the paper goes on with a description of the main operations supported by the WSIGS. Before concluding the paper, a number of more advanced features and future work are finally discussed.

### 1.1 Web Service Technology

Since the advent of the World Wide Web, it has been evident that the prevailing mode of human-to-machine interaction would ultimately be supplemented and extended by technology designed to support direct machine-to-machine interaction, sometimes driven by humans and sometimes in a completely autonomous mode. Recently this has begun to be manifested in the form of Web services. Web services are essentially some captured application or business logic that is programmatically encoded to execute on a Web server by exposing its functional capabilities as methods available to clients over HTTP. Each method can therefore be published as a URL, can accept parameters and return data, typically encoded in XML. The W3C defines a Web service as "... a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [8].

The encoding and protocol stack typically used for creating and publishing Web services includes the following technologies: WSDL, SOAP and UDDI<sup>2</sup>. WSDL is the standard means for expressing Web service descriptions. SOAP is a protocol defining the exchange of messages containing Web service requests and responses. UDDI is the directory services schema commonly used to register and discover Web services.

Web service technologies are now impacting a broad range of commerce and industry. This is not because they are based on any especially innovative technology, but rather because they are a relatively straightforward means of constructing machine-to-machine relationships using simple and proven technology. In addition, Web services are fast approaching

\* 0-7803-8566-7/04/\$20.00 © 2004 IEEE.

<sup>1</sup>JADE is the Java Agent Development Environment. Additional information can be found at <http://jade.tilab.com>

<sup>2</sup>WSDL: <http://www.w3c.org/TR/wsdl>,  
SOAP: <http://www.w3c.org/TR/soap>, UDDI: <http://www.oasis-open.org/>

the point of critical mass beyond which they are likely to become an accepted mainstream technology. Tracing this path through the next few years we can predict that emerging technologies, such as the Semantic Web [17] will substantiate these trends to take the basis of Web services towards powerful modalities that employ semantic expressivity, autonomy and knowledge processing, as highlighted by Davies et al. in [4].

A Web service should be used when an application builder wishes to expose some reactive operation expressible as a programmatic function with or without parameters that may or may not return a response. This is essentially a remote method invocation using an XML encoded message over HTTP.

## 1.2 Agent Technology

Separately and in parallel to the emergence of Web services the field of multi agent systems<sup>3</sup> has been focusing on providing support to the conceptual and programmatic encapsulation of autonomous, goal-driven behaviour for a wide variety of applications and software solutions. Many of these ideas are captured in the FIPA specifications, such as [5], which define broadly accepted standards for building openly interoperable agent-based systems. Similar to Web services, software agents can also be used to encapsulate some business or application logic, but differ in that they do not simply expose functionality as methods over a fixed protocol. Rather, software agents can offer multiple services, or behaviours, that can be processed concurrently<sup>4</sup> and activated by specifying goals. A goal, instead of identifying a particular method to be invoked, states an abstract objective to be achieved using whatever resources the agent has available, including knowledge and action behaviours. By these means a software agent is capable of making proactive and autonomous decisions about how to act in a given situation.

Multi agent systems have been the subject of massive amounts of research in recent years and are beginning to find their way into commercial applications [13]. An exhaustive overview is beyond the scope of this paper, but essential pointers include [18] and [10]. Probably the most pervasive agent system in use today is JADE, especially in the context of research driven applications. JADE is an open source, FIPA<sup>5</sup> compliant agent platform designed to be a middleware solution for developing and executing peer-to-peer applications based on the software agent paradigm. In particular, JADE is specifically designed for creating applications that can seamlessly interoperate across fixed and wireless environments. Even though the WSIGS is intended for use with the JADE system, it can also work seamlessly with any agent system exposing FIPA compliant interface bindings.

An agent should be used when an application builder wishes to create a programmatic entity capable of deciding

for itself, based on its knowledge, state and capabilities, the best way to solve a given problem, or goal. Communication between agents is typically loosely-coupled allowing agents to process service requests concurrently and asynchronously.

## 1.3 Agent and Web Service Integration

Integrating Web services and software agents brings about the immediate benefits of connecting application domains hosting one or the other technology: A Web service should be able to invoke an agent service and vice versa. However, once this interconnection is established software agent concepts and technologies will help enable new and advanced operational and usage modalities of Web services. Several arguments have been made to support this case, including [15] [12] and [11], but perhaps none more evocative than statements made in the Web Services Architecture specification of the W3C [8] which clearly expresses the notion that, "... *software agents are the running programs that drive Web services - both to implement them and to access them as computational resources that act on behalf of a person or organization*".

In this perspective, identifying a means of connecting agents and Web services is the motivation and central foundation of our work. Due to the evident technology mismatches between Web services and software agents, including strong vs. loose coupling and representational encodings, we have identified an approach that introduces an intermediary service entity, the WSIGS, which is designed to encapsulate the functionality required to connect the two domains, whilst ensuring minimal human intervention and service interruption.

From the perspective of agents, Web services are simply programmatic entities that can be called upon to perform an advertised and typically unitary function. To consumers of Web services, agents can form a powerful means of indirection by masking the Web service for purposes of redirection, aggregation, integration or administration. *Redirection* describes the case where a Web service may no longer be available for some reason, or the owner of the Web service wishes to temporarily redirect invocations to another Web service without removing the original implementation. *Aggregation* allows several Web services to be composed into logically interconnected clusters providing patterned abstractions of behaviour that can be invoked through a single service interface. *Integration* describes the means of simply making Web services available to consumers already using, or planning to use, agent platforms for their business applications and *Administration* covers aspects of automated Web service management where the agent autonomously administers one or more Web service without necessary intervention from a human user. From the opposite perspective the architecture proposed herein closes the loop by enabling Web service clients to invoke application services exposed by agents. In this instance, the type of agent service made available is naturally restricted due to the limited expressivity of typical service invocation calls initiated by Web service clients. The intrinsic

<sup>3</sup>Also known as software agents or intelligent agents.

<sup>4</sup>Typically either preemptively or non-preemptively according to implemented thread models.

<sup>5</sup>Foundation for Physical Intelligent Agents, <http://www.fipa.org>

implication here is though, that applications are now able to bidirectionally seamlessly bridge the agent and Web service domains.

Three key features of the WSIGS are:

- *Transparency.* The WSIGS is designed to be operationally transparent to invoking entities, whether they be agents or Web services. For example, an agent wishing to invoke Web service sends the invocation message directly to the WSIGS with a property of the message content set to the name and address of the Web service. The WSIGS will seamlessly transform the message, issue it and transcribe any response before returning it to the invoking agent.
- *Automation.* Once operational, the WSIGS requires no manual intervention or configuration.
- *Integration.* The WSIGS encapsulates and integrates the functionality required to deliver the above features without requiring additional external resources.

## 2 Related Work

The work reported in this paper is an attempt to identify a relatively simple, yet extensible model that brings about loosely-coupled integration of agents and Web services. As such it is strongly influenced by and indeed evolved from, several existing contributions to the field such as that reported by Lyell et al. [11] which discusses the concept of a hybrid J2EE/FIPA-compliant software agent system using Colored Petri Nets. This approach identifies two key concepts that (1) agents should be able to expose their services as Web services for the potential use of non-agent clients and (2) these *Web service-enabled* agents should advertise using both a UDDI registry and a FIPA Directory Facilitator (DF). Our approach also addresses these issues, but without the specific use of J2EE or special “Web service agents” to expose Web services. Instead the WSIGS is intended for use with any J2\*E technology platform and deliberately avoids the introduction of specialized agents for handling the exposure of Web services. Instead, the WSIGS is an independent set of codecs that can process and translate Agent messages and Web service calls without bringing any agents or Web services within the Gateway boundary. We find that this approach is more scalable than ‘one agent for one Web service’, as only new entries in the WSIGS DF and UDDI repositories need be generated each time a new agent service or Web service is exposed - rather than creating a new agent for the purpose.

Also in [14] we are shown how automation can be driven by agent generators reasoning about the semantic descriptions of the Web services they are configured to compose and use. However, perhaps the most directly referential work relating to the WSIGS architecture is that of the Gateway Model produced by the Agentcities.NET<sup>6</sup> Web service working group.

<sup>6</sup>Agentcities.NET is a European Commission funded 5th Framework

The recommendation produced by this group [7] details a two-part solution to constructing a bidirectional agent-to-Web service Gateway. The model, intended for use by the JADE agent system, consists of two separate implementations that (to date) have yet to be integrated into a unitary solution. One aspect, called WSDL2JADE [7] is designed as a wrapper solution for converting agent sourced ACL encoded requests into the appropriate Web service operation calls and corresponding responses back into ACL. The tool generates a JADE proxy agent for an existing Web service described by a WSDL file, making it possible to call Web services indirectly within an agent environment. The second aspect, called WSAI [7] and also implemented using JADE, allows an agent service to be published as a Web service. The implementation consists of a Web Services Agent Gateway (WSAG) and an agent Generator. The WSAG manages the transition from agents to Web services and the Generator is a support tool for generating agents that operate within the Gateway providing a concrete Web service interface for a particular external agent. When a Web service invokes a SOAP call on the Gateway, the Gateway transforms this synchronous call into an asynchronous message communication via FIPA ACL messages to the software agent that provides the service. In its current form, this solution is neither integrated, transparent nor automatic and has the additional drawback of requiring a degree of manual configuration. In addition it has limited support for extensions enabling advanced services such as proxying, composition and semantic filtering. The WSIGS approach takes some of the ideas generated by this project and extends them to form a model of a transparent, automatic and extensible Gateway service.

Regarding some of the advanced features discussed in the latter half of this paper, we note that [12] discusses the use of agents as proxies that assist in selection of Web services according to the quality of matching criteria. The approach supports the dynamic selection of services as a path toward autonomic computing where computational resources are self-managing and self-configuring. In addition, in [1] and [16] the authors discuss the use of process description languages for enacting business processes in the contemporary workplace. They note that strict adherence to prescribed workflows implies that systems are largely unable to adapt effectively to unforeseen circumstances. The work proposes that workflow description languages be used to specify multi agent systems, specifically advancing the idea that the Business Process Execution Language for Web Services [3] can be used as a specification language for expressing the initial social order of a multi agent system, which can then intelligently adapt to changing environmental conditions.

IST project, IST-2000-28384, completed in 2003, aiming at creating an open dynamic service environment based on (FIPA compliant) agent technology and make it accessible to potential users. More information can be found at <http://www.agentcities.org/EUNET/>

### 3 Architectural Model

The WSIGS is a stand alone, encapsulated service that provides transparent, bidirectional transformations between FIPA compliant agent services and Web services that use the standard WSU<sup>7</sup> stack. We have chosen to design a solution that complies with FIPA both due to its status as a widely adopted industry standard and also as we wish to provide the WSIGS as a service available to the JADE/LEAP<sup>8</sup> platform, which is indeed FIPA compliant. With respect to Web services, to ensure some baseline compliance we employ the WSU stack, also due to widespread industry adoption.

#### 3.1 Assumptions

The following assumptions were made when designing the WSIGS architecture:

- All agents are assumed to be FIPA compliant and capable of communicating with FIPA-ACL encoded messages.
- All Web services operate using the standard Web service stack consisting of WSDL for service descriptions, SOAP for message encoding and UDDI for directory services.
- The WSIGS is designed as a platform service accessible to both agents running on and external to JADE platforms, and Web service clients operating from any accessible network locations.
- The WSIGS is registered as an agent service in FIPA Directory Facilitators and as a Web service endpoint in UDDI directories (the default identifier for the WSIGS is *sig*).
- All interactions between the WSIGS and agents use ACL encoded FIPA-Request and FIPA-Inform performatives [6]. This simplifies the operation of the WSIGS and are essentially all that is needed to invoke basic request-response Web services.
- All ontologies used by agents are published and available to the WSIGS in a machine readable form.

#### 3.2 Architecture

The problem as specified is to provide an automatic means of mapping the functional and representational dependencies associated with the invocation of agent services onto those associated with Web services and vice versa. Our proposed solution introduces an intermediary processor into the communication path between these service calls. The WSIGS allows both agent and Web services to register with it and thereby publish their service descriptions to consumers outside their normal operational domain. The WSIGS will then intercept calls to these registered services allowing agents to

invoke Web services and vice versa by transforming message encodings and creating service access endpoints. A functional overview of the WSIGS is described by Figure 1.

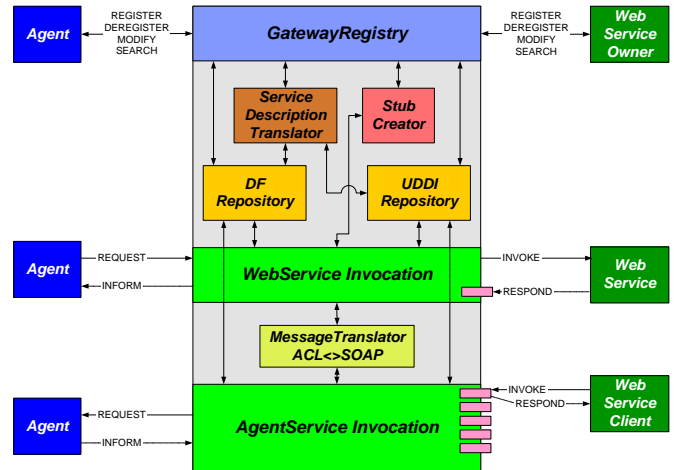


Figure 1: Functional overview of the WSIGS architecture entities.

As illustrated, the WSIGS contains several operational components, including a FIPA compliant Directory Facilitator (DF) and a Web service stack compliant UDDI repository. These internal registries are at the heart of the WSIGS, maintaining records of all agent and Web service descriptions that have been registered with, and can be invoked via, the Gateway. Access to these registries is provided to both agents and Web service clients by exposing appropriate interfaces providing the standard operations:

- Registration of a new service description.
- Deregistration of an existing service description.
- Modification of an existing service description.
- Searching for a registered service description.

While the semantics of these operations are described further in Section 4, the remaining operational components are described below.

#### The GatewayRegistry

This is the logical component that receives and processes incoming directory operations, thereby controlling access to the internal registries. ACL encoded service descriptions received from agents are stored in the internal DF registry and automatically translated (by the ServiceDescriptionTranslator component) into their WSDL equivalent and stored in the internal UDDI registry. The reverse case is also true for WSDL encoded service descriptions received from Web service owners. In this manner the WSIGS maintains a mapping of each service description in both ACL and WSDL forms. Additionally, a RegistryID tag with the same unique identity

<sup>7</sup>WSDL, SOAP and UDDI

<sup>8</sup>LEAP is the Lightweight Extensible Agent Platform, a version of JADE for mobile devices and handsets

is assigned to the DF and UDDI instances of the service description to ease future lookups across both directories. In the case of malformed service descriptions an exception is raised, i.e., syntactic validation, and treated accordingly relative to the sender type (agent or Web service).

### The ServiceDescriptionTranslator

Utilized by the GatewayRegistry, the ServiceDescriptionTranslator component, shown in Figure 2, acts as a transformation filter with two paths of operation.

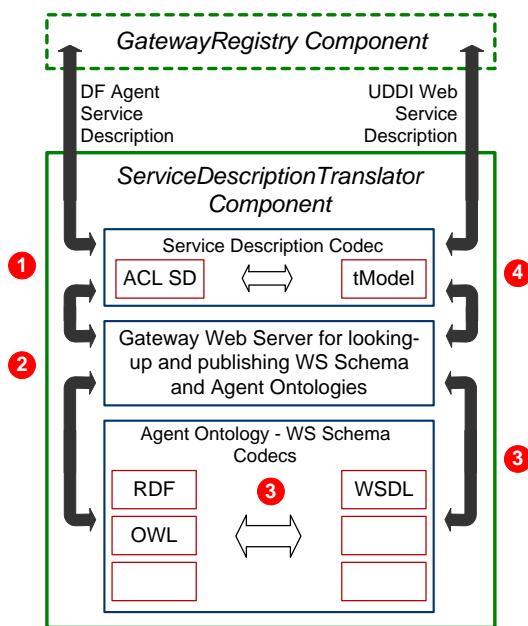


Figure 2: Internal operation of the ServiceDescriptionTranslator component.

The first path is initiated when a request to register a new ACL encoded agent service description [5] is received by the GatewayRegistry component. In this instance the following sequence of events takes place, as indicated in Figure 2.

1. The ACL service description specified in the agent registration message is passed to the Service Description Codec for transformation into the corresponding UDDI tModel [2] to be returned to the GatewayRegistry for storage in the Gateway UDDI directory.
2. The *ontologies* attribute of the ACL service description should contain a published URL referencing the ontology used to describe the agent service. This reference is passed to the Gateway Web server which will resolve the reference and pass the resulting ontology to the Agent Ontology - Web service Schema Codec.
3. The Agent Ontology - Web service Schema codec(s) translate the referenced ontology into an equivalent WSDL schema and returns the result to the Gateway

Web server where it is published with a reference for use when building the service tModel. The Gateway can plug-in any suitable codec including RDF-WSDL and OWL-WSDL.

4. The tModel is returned to the GatewayRegistry containing a reference to the newly published WSDL service description.

The second path of operation is the inverse case of the above description, i.e. tModels are translated into ACL service descriptions and WSDL is translated into RDF/OWL/etc. ontologies according to configurable preferences. Resulting ontologies are published to the Gateway Web server for use by agent systems.

This process ultimately ensures that all registered service descriptions are duplicated across both internal registries.

### The StubCreator

Web services are typically exposed as service endpoints, herein defined as stubs, which are called as remote invocations. Thus whenever a new agent service (that is to be exposed to Web service clients) is registered with the WSIGS, the GatewayRegistry invokes the StubCreator which generates a stub to be exposed via the AgentServiceInvocation component. The stub is built from the WSDL service description of the agent service held in the WSIGS UDDI directory and generated by the ServiceDescriptionTranslator component as previously described. When an invocation is received from a Web service or Web service client, the stub is activated and manages reception and processing of the incoming message and any subsequent conversation context. In addition, whenever an agent invokes a Web service via the WebServiceInvocation component, if a response is expected from the Web service a temporary stub is created by the StubCreator. This stub is removed once the expected reply is received.

### WebServiceInvocation

Described by Figure 3, when an agent wishes to invoke a Web service it first looks up the ACL encoded service description from the WSIGS's DF directory (translated from the original WSDL by the ServiceDescriptionTranslator). Using this, the agent formulates a standard FIPA-Request message containing the Web service invocation call and sends it to the WebServiceInvocation component of the WSIGS which is exposed as a standard FIPA message transport endpoint. This component then uses the ServiceDescriptionTranslator component once again to map the ACL encoded service description into the corresponding WSDL, stored in the internal UDDI directory. Finally, the returned WSDL description is used to create a SOAP encoded message using the MessageTranslatorComponent which contains a bi-directional codec for transforming ACL into SOAP and vice-versa. This message is then used to invoke the appropriate Web service with a return stub being exposed to receive any expected response. Any message received by this

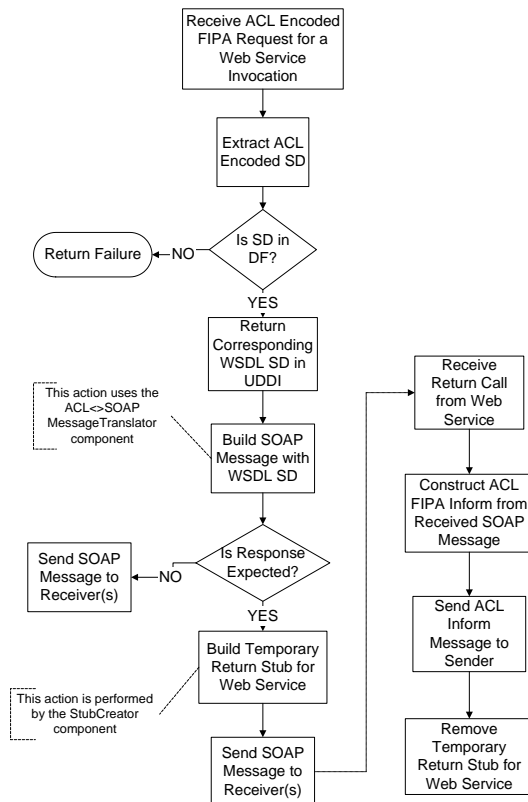


Figure 3: Web service invocation.

stub is translated back into ACL and returned to the calling agent using a FIPA Inform message with any reply-specific attributes from the original ACL service invocation call.

### AgentServiceInvocation

This component performs the inverse of the previous process with one significant difference. For a Web service client to invoke an agent service it must be able to make a SOAP encapsulated call onto the agent service, or a stub that reflects the functionality of that agent service. To this end, the WSIGS creates and exposes a service stub for any agent service that registers with it. This stub is created using the corresponding WSDL description of the agent service held in the Gateway UDDI directory presenting a service binding to external Web services. When an invocation is received from a Web service, the stub first looks up the WSDL service description in the internal UDDI directory, using the ServiceDescriptionTranslator to identify the appropriate ACL encoded service-description stored in the local DF. From this a FIPA Request message is constructed containing the service call and parameter values received from the calling Web service client. The message is then sent to the agent exposing the agent service. Additionally, the WSIGS keeps track of all ACL messages sent in a conversation context. The receiving agent processes the ACL message and assuming the absence of errors, will invoke the correct agent service and

return any expected results to the WSIGS as a FIPA Inform message. Inside the WSIGS, this message is parsed using the MessageTranslator the return value passed back to the calling Web service client through the existing agent service stub interface. If a response is expected from the agent service and none is received within a specified period, then a timeout exception can be raised.

### The MessageTranslator

This contains the parsing functionality for transforming SOAP encoded messages into ACL encoded messages and vice-versa. In both directions parse-trees are constructed from the incoming message, which can then be traversed with each leaf node being translated into the corresponding encoding. This requires a schema that defines the SOAP equivalents of ACL slots and attributes, and vice versa.

## 4 The WSIGS in Action

The WSIGS has two principal modes of operation: an agent service invoking a Web service and a Web service invoking an agent service. Ancillary to these are the operations associated with registering services with the WSIGS. Additional advanced features are discussed in Section 4.2.

### 4.1 Standard Operation

The standard features of the WSIGS are described by the following operations:

#### WSIGS Initialisation

When initializing, the WSIGS registers with any available or pre-stated external registries in both the agent and Web service domains. From the agent perspective the WSIGS exposes the same interfaces (register, deregister, modify and search) as that of a typical FIPA compliant DF. From the Web service perspective, the WSIGS exposes the same functionality, but encoded as UDDI endpoints as described by the UDDI Schema [2].

As the WSIGS is a Gateway all requests for service (and responses from invoked services) must be sent to the WSIGS with the identity of the ultimate receiver encoded as a property of the request/response message.

#### Register a Service Description

Following the process described by Figure 4 the service descriptions of agent services and Web services are submitted to the WSIGS by their corresponding owners. Once received by the WSIGS, a description is automatically translated into either ACL if received in WSDL form via a UDDI registration or the inverse if received via a DF registration. These translation processes are performed by the ServiceDescriptionTranslator component and consist of several steps as described in Section 3.2.

The dual descriptions are stored in the internal registries with a common RegistryID and thereby made available for discovery by external entities. Finally, if registering an agent service the stub interface used by a Web service to invoke it

is exposed via the AgentServiceInvocation component of the WSIGS.

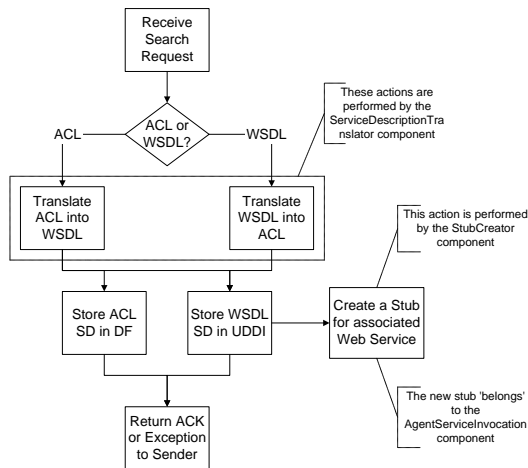


Figure 4: Registering an agent or Web service with the WSIGS entities.

### De-register a Service Description

A de-registration request received by the WSIGS will remove both local copies (DF and UDDI) of the service description and any stub established for Web service sourced invocations.

### Modify a Service Description

A modification request received by the WSIGS will replace an existing service description. This procedure can be treated in the same way as a new Registration.

### Search for a Service Description

Unlike the previous operations, a search request received by the WSIGS will act only on the appropriate internal registry; DF in the case of agent services and UDDI in the case of Web services.

### Invoke an agent service from a Web service client

In terms of exchanged messages, a typical scenario might be a Web service client initiating a SOAP message to invoke a `getTime` service previously published with the WSIGS by an agent resident in a JADE platform. We assume that this service has been successfully registered with the WSIGS, that a stub has been exposed and that the Web service client has searched the WSIGS UDDI repository for the appropriate service description. Once a SOAP message carrying a call to invoke the service is received by the WSIGS it is processed and a corresponding FIPA Request message created and sent to the agent hosting the `getTime` service. The result is then sent back by the agent as a FIPA Inform message to the WSIGS, transformed into a SOAP message and returned to the initiating Web service client.

### Invoke a Web service from an Agent

Similar in form to the previous case, the inverse invocation direction is initiated by an agent sending a FIPA Request message to the WSIGS specifying the service description and parameters of a call it wishes to make onto a Web service. Assuming that the required Web service is registered, the WSIGS transforms the ACL message into the corresponding SOAP encapsulated WSDL and sends this to the appropriate Web service. If a reply is expected from the Web service a stub is exposed by the WSIGS to receive it. In such a case, upon reception of a reply message, it is transformed into a FIPA Inform message and returned to the initiating agent.

## 4.2 Advanced Features

Beyond the standard features offered by the WSIGS we anticipate that several more advanced features will become available. These relate to ways in which the WSIGS (according to security constraints) can intercept and manipulate the basic service invocation calls, such as proxying, redirecting and composing. However, all of these features are intrinsically dependent on the level of semantic expressivity present in service descriptions, which in the current model is quite low. We are currently investigating the use of OWL-S<sup>9</sup> to add semantic depth to service descriptions and so provide more opportunities to apply reasoning to the selection and manipulation of services [9].

### Redirection

An additional operation offered by the WSIGS is the ability to perform automatic fail-over redirection when a service published by the WSIGS is no longer unavailable. This can be managed pre-execution by notifying the WSIGS (via a reserved message type) of which service(s) should be substituted in the event of a failure condition. Alternatively, if no pre-selection has been made the WSIGS can be configured to dynamically search its internal registries for a suitable replacement service. The effectiveness of the latter solution is dependent on the level of semantic detail used for the service descriptions and is best delivered through a specialised proxy component.

### Proxying

During standard operation the WSIGS acts passively by exposing access endpoints via the conventional registry and invocation components. However, extending the redirection model the WSIGS could house an internal proxy mechanism designed to interface between services whose descriptions are held by the WSIGS and their invoking clients [12]. The proxy can contain rule engines or reasoners capable of playing an active role in selecting a service according to qualitative matching of application criteria with published service descriptions. To be effective the descriptions would need to be published in a representation such as DAML-S or OWL-S.

<sup>9</sup>The Ontology Web Language Service ontology, details available at <http://www.daml.org/services/owl-s/1.0/>

## Workflow and Composition

There are several complete, ongoing and emerging initiatives that are defining extended notations for the management and operation of Web services. To name but a few there are: WS-Addressing, WS-Attachments, WS-Context, WS-Coordination, WS-Federation, WS-ReliableMessaging, WS-Routing, WS-Transaction, WS-Trust, XML-DSig, XML-Encryption, XKMS, SAML, XACML, ebXML, WSBPEL and WSRP. Amongst these are several compositional notations that express the flow of control and data across a collection of Web Services whose choreography performs a workflow. Specifically, the WSBPEL<sup>10</sup> specification enables businesses to employ workflow management that enacts business processes described by the language [16].

However, strict adherence to prescribed workflow makes it impossible for a system to adapt to unforeseen circumstances and it is anticipated that agent systems could act as the intelligent controllers of the workflow enactment mechanism. The WSIGS is well positioned to enable this, and dynamic composition of Web services, by providing connectivity between the two domains.

## 5 Conclusions

This article reports on the design of a Gateway architecture for enabling transparent, automatic connectivity between Web services and agent services. It is anticipated that such systems will help prevent the risk of an asymptotic or divergent relationship developing between these two complementary technologies. The proposed architecture, termed the Web Service Integration Gateway Service (WSIGS) contains several components that operate on internal registries to maintain records of all registered services (both agent and Web based). These services can then be seamlessly invoked via the Gateway, whether it be an agent service invoking a Web service or vice versa. A prototype of this model is currently in development, with the finished product intended to become an add-on component of the JADE system in a planned future release.

In the latter part of the paper we discuss some advanced features of the WSIGS. We expect that the initial process of creating such a Gateway service bridging agents and Web services is only a prelude to the potential it offers as a means to compose, administer and manipulate service populations.

## References

- [1] P. Buhler, J.N. Vidal, and H. Verhagen. Adaptive workflow = web services + agents. In *Proc. of the International Conference on Web Services*, pages 131–137, Las Vegas, U.S.A., July 2003. CSREA Press.
- [2] OASIS Consortium. *UDDI Specifications and Schema*. <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>, 2004.
- [3] OASIS Consortium. *Web Services Business Process Execution Language Specification*. <http://www.oasis-open.org/committees/>, 2004.
- [4] N.J. Davies, D. Fensel, and M. Richardson. The future of web services. *BT Technology Journal*, 22(1):76–82, January 2004.
- [5] Foundation for Intelligent Physical Agents. *FIPA Agent Management Specification*. <http://www.fi.pa.org/specs/fipa00023/>, June 2002.
- [6] Foundation for Intelligent Physical Agents. *FIPA Communicative Act Library Specification*. <http://www.fi.pa.org/specs/fipa00037/>, June 2002.
- [7] Agentcities Task Force. *Integrating Web Services into Agentcities Recommendation*. <http://www.agentcities.org/rec/00006/actf-rec-00006a.pdf>, 2003.
- [8] W3C Web Service Architecture Working Group. *Web Service Architecture Recommendation*. <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>, 2004.
- [9] M. Laukkanen and H. Helin. Composing workflows of semantic web services. In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia, July 2003.
- [10] Michael Luck, Peter McBurney, and Chris Preist. *Agent Technology: Enabling Next Generation Computing*. <http://www.agentlink.org/roadmap>, 2003.
- [11] M. Lyell, L. Rosen, M. Casagni-Simkins, and D. Norris. On software agents and web services: Usage and design concepts and issues. In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia, July 2003.
- [12] E.M. Maximilien and M.P. Singh. Agent-based architecture for autonomic web service selection. In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia, July 2003.
- [13] M. Rajdou. Software agents in business: Steady adoption curve. Technical report, Forrester Research, USA, 2003.
- [14] D. Richards, S. van Splunter, F. Brazier, and M. Sabou. Composing web services using an agent factory. In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia, July 2003.
- [15] K. Sycara, M. Paolucci, and J. Soundry N. Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. *IEEE Internet Computing*, 8(3):66–73, May 2004.
- [16] J.M. Vidal, P. Buhler, and C. Stahl. Multiagent systems with workflows. *IEEE Internet Computing*, 8(1):76–82, January/February 2004.
- [17] W3C. *The Semantic Web*. <http://www.w3.org/2001/sw>, 2004.
- [18] Michael Wooldridge. *An Introduction to MultiAgent System*. Wiley and Sons, 2002.
- [19] Collected Works. *EXP: Special Issue on JADE*, volume 3:3. Telecom Italia Laboratories, September 2003.

<sup>10</sup>The Web Services Business Process Execution Language, formerly known as BPEL4WS